Applied Kernel Density Estimation: Dynamic Spatiotemporal Analysis of Density Maps on Crime Data

David G. Smalling

Abstract The task of identifying meaningful patterns in spatiotemporal datasets lies at the heart of data mining. This paper outlines the development of a system which utilizes latitude and longitude data of criminal acts to generate an image sequence of density maps which are used in the construction of a video of criminal movement within a study region. This paper provides a methodology for a detailed study of crime movement in New Haven, Connecticut and Kingston, Jamaica, in which the effects of topographical changes and sociological trends such as political upheavals and placement of landmarks are analyzed. In addition the attempt is made to quantify Spatiotemporal Correlation between multiple clusters and develop algorithmic heuristics relative to the dataset in order to reduce the computational complexity of density map generation.

David G. Smalling
Yale University, Department of Computer Science, USA
david.smalling@yale.edu

1 Background

In Data Mining and Statistics the Kernel Density Estimation (KDE) is a method of approximating the probability density function of a random variable over specific domain. In recent years KDE has proven to be an indispensable tool in spatial analysis because of its effectiveness in pinpointing “hot spots” in the point data, which are locations of comparatively elevated density compared to the rest of the study region. O’Sullivan et al believe that KDE is “one of the most useful applications in applied Geographic Information System Analysis” *(O’Sullivan et al., 2003).* Recent work done by the University of Ljubljana demonstrated methods by which physical laws may be extracted from experimental data with unprecedented accuracy. Kernel Density Estimation is also often used by crime prevention agencies such as the Federal Bureau of Investigation (FBI) to analyze subtle trends in criminal movement and to predict the movement of repeat offenders. The U.S. Department of Education has utilizes a software package under a program called Project Strike Back student financial aid records for the FBI. It sieves out the names of suspected terrorists through the Education Department's database of information, which is derived from students who fill out the Free Application for Federal Student Aid (FAFSA).

Prof. Parris Lyew Ayee of the University of the West Indies in Kingston Jamaica has made the provision of a dataset containing very detailed accounts of violent (non-fatal) crimes in Kingston, Jamaica for the year 2004. Professor Ayee is himself an expert in spatial modeling and analysis. After having completed groundbreaking research at the University of Oxford in Geographic Information Systems, he has supervised numerous

studies for the Jamaica Constabulary Force and has significantly optimized the positioning of police efforts in Jamaica.

2 Design Framework

Definition 2.1 Frame A frame of size *f* is a density map derived from a collection of latitude-longitude data from *f* different rows.

The traversal of the aforementioned frame through the dataset is at the underpinning of this method of analysis. For a specified collection of data points in a Euclidean environment such as latitude-longitude data, we take data from the m^{th} row to the $(m+f-1)^{th}$ row and derive a density map corresponding to *f* elements inclusive. The construction of the necessary image sequence involves continuing this process, taking data from $(m+1)^{th}$ row to the $(m+f)^{th}$ row up to the frame spanning $(D-f+1)^{th}$ row to the D^{th} row. This technique is depicted graphically below. Where $D = \dim_{row}(dataset)$.

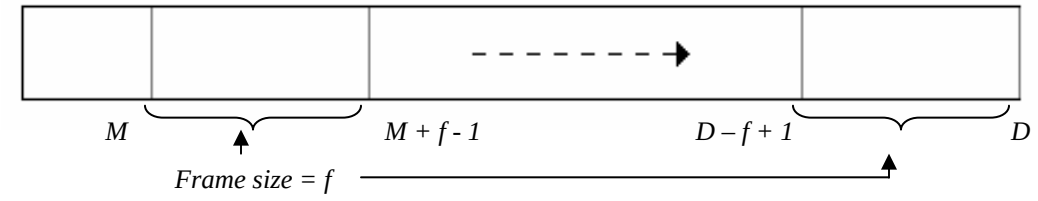

Figure 2.1

3 Algorithmic Framework

(Note: The MASS package is required to execute one or more of the following algorithms)

Algorithm 3.1 Category list. The *category_list* algorithm takes as its arguments a data set and an integer. Returns the unique elements in the column corresponding to the integer given.

```
category_list = function(data,col){
        unique(data[,col])
}
```

Definition 3.1 Filter. A filter is constructed as a list of the form $(x, n_1, n_2, \ldots, n_m)$ where the elements of the list are integers subject to the constraints:

$$0 < x < \dim_{col}(dataset) + 1$$

$$0 < n_i < \text{length}(\text{ category\_list}(\ dataset\ , x\ )\ )$$

The x term in the filter corresponds to the column to which the filtering elements belong and each n_i corresponds to the location of the desired filtering element within the data set's category list. The aforementioned constraints on the values of x and n_i follow trivially from their definitions.

Definition 3.2 Filtering Buffer Sequence. The buffer sequence of a filter consists of a list of integers corresponding to the rows of the dataset which satisfy the conditions of the said filter.

Algorithm 3.2 Buffer Sequence Assembler. The *build_buffer* algorithm takes as its arguments a data set and a filter and returns the corresponding Filtering Buffer Sequence.

```
build_buffer = function(data,filter){
        i = 1 #iterating term
        D = dim(data)[1]
        buffer = c() #sets the buffer sequence NULL

        key = as.numeric(filter[1]) #defines the column containing the filtering elements

filter_list = category_list(data,key)[filter[-1]] #builds a raw representation of the filtering elements using
the numerical representation stored in the filter.

#the each iteration of the following loop we seek to check if the element in the i^th row of the filtering
column satisfies the conditions. To achieve this we perform a comparison between the element and each
member of the filter list then reduce the resulting truth table to unique element form. By then sorting the
elements in reverse alphabetical order it suffices to check if the first element is TRUE since there are only 3
possibilities for the resulting list ( [TRUE], [TRUE, FALSE] or [FALSE] ).

while(i <= D){
        if(sort(unique(data[i,key] == filter_list), decreasing = TRUE)[1] == TRUE){
                buffer = c(buffer,i)
                                        }
                i = i + 1
        }
return(buffer)
}
```

Algorithm 3.1 Image Sequence Rendering Device. The primary image rendering algorithm takes as its arguments (in order):

- Frame size
- The Data set,
- Data set name
- Filter
- Column containing the longitude data
- Column containing the latitude data

```
bmp_stream = function(f,data,R,title,filter,long_col,lat_col){
        i = 1
        D = dim(data)[1]
        lim = D - f + 1 #this is a lower boundary limiting  term for the export loop.
        format = ".bmp"
if(filter[1] != 0){
        buffer = build_buffer(data,filter)
        buffer_lim = length(buffer) - f + 1
} #defines our default for the filter as 0 in which case all data is taken in the generation of each frame
#when the bmp_stream function is invoked, one of the two filtering subroutines is called based on whether
a filter is being imposed on the data.
##Unfiltered Export
if(filter[1] == 0){
        while(i <= lim){
        Frm <- kde2d(data[i:(i+f-1),long_col],data[i:(i+f-1),lat_col], n = R) #performs density estimation
                name <- paste(title,"-",i,"-",R,format,sep="")  #creates filename
                bmp(file = name, bg="transparent") #activates bitmap device
                image(Frm) #displays image to local bitmap device
                dev.off() #deactivates bitmap device
                i = i+1 #increments frame lower limit
        }
}
##Filtered Export # the only difference in the filtered export subroutine is that the f elements for the are
derived from the filter's buffer sequence by means of buffer[i:(i+f-1)].
if(filter[1] != 0){
        while(i <= buffer_lim){
        Frm <- kde2d(data[buffer[i:(i+f-1)],long_col],data[buffer[i:(i+f-1)],lat_col], n = R)
        name <- paste(title,"-",i,"-",R,format,sep="")
                bmp(file = name, bg="transparent")
                image(Frm)
                dev.off()
                i = i+1
        }
}
return(i) #return the final frame's lower bound index, for error correction and scalability purposes only.
}
```

4 Calibration

In order to accurately pin the generated density maps to the locations, a plot is exported that contains the bounds of our study region and some sample points. These points are then loaded into a map system with verified accuracy (eg: Google Earth). Using these images, accurate scaling of the density plot to the map of the study region may be achieved.

Such calibration was performed on the New Haven Crime data using the following code. *(Note that the sample points used are the first two points in the data set).*

```
long_min = min(mydata[,10])

long_max = max(mydata[,10])

lat_min = min(mydata[,11])

lat_max = max(mydata[,11])

calibrate = matrix( c(long_min, lat_min, long_max, lat_max, long_min, lat_max,
long_max, lat_max, mydata[1,10:11], mydata[2,10:11]), ncol=2, byrow=TRUE)

plot(calibrate)
```

The resulting matrix (Calibrate) is:

```
[1,] -72.9868  41.2477
[2,] -72.851   41.3505
[3,] -72.9868  41.3505
[4,] -72.851   41.3505
[5,] -72.9353  41.3279
[6,] -72.9347  41.2923
```

A plot of the matrix yields the following overlay:

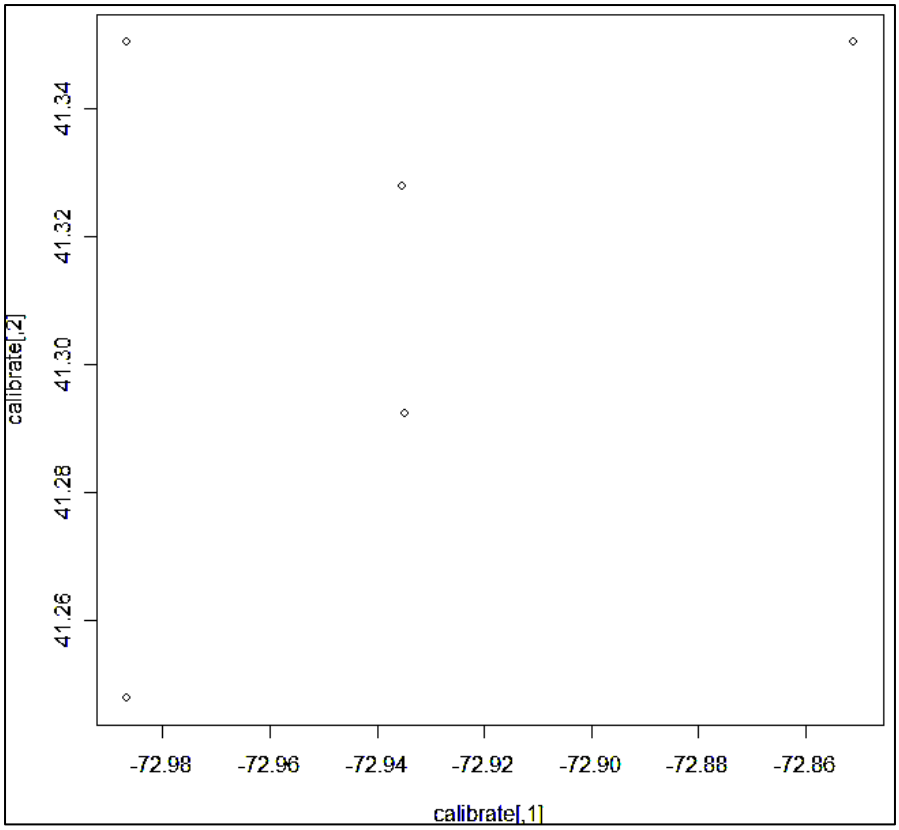

Figure 5.1

Figure 5.1

5 Notes

This method of analysis proves useful in the study of non-organized crime environments and observation of aggregate movement of petty crimes that are unplanned and usually quasi-random in nature.

Due to the computational complexity of the algorithms described above it was difficult to run comprehensive tests on the datasets.